# 2do Parcial – 2017s2 - Estructuras de Datos – UNQ

***Aclaraciones:***

- *Esta evaluación es a libro abierto. Se pueden usar todas las funciones y tipos de datos vistos en la práctica y en la teórica, salvo que el enunciado indique lo contrario.*
- *No se olvide de poner nombre, nro. de hoja y cantidad total de hojas en cada una de las hojas.*
- ***Deje un ESPACIO VACÍO de 10cm al principio de la 1era hoja del examen.***
  ***PRESTE ATENCIÓN. ES IMPORTANTE.***
- *Le recomendamos leer el enunciado en su totalidad y organizar sus ideas antes de comenzar la resolución. Dedique el tiempo necesario para entender la estructura planteada antes de comenzar a resolver.*
- *Recuerde que la intención es medir cuánto comprende usted del tema. Por ello, no dude en escribir todo lo que usted tiene en cuenta para resolver un ejercicio, en probar sus funciones con ejemplos, etc.*

## Organizador de autores de software

El objetivo de este parcial es implementar un organizador que almacena información sobre los autores de distintos programas. Los programas del organizador se identifican unívocamente mediante un Código Identificador (Checksum), generado automáticamente de forma externa (para nosotros esto será transparente, pero si usted sabe lo que es una función de checksum puede imaginarse cómo).

Trabajaremos con dos TADs auxiliares: el tipo abstracto `Checksum` para representar códigos identificadores de programas, y el tipo abstracto `Persona` para representar programadores. Las únicas operaciones que nos interesan de estos TADs son la comparación por igual y la comparación por mayor (o sea, ambos tipos se pueden comparar por igualdad (`Eq`) e incluyen un orden (`Ord`). El resto de las operaciones de estos tipos no son relevantes a esta evaluación.

El tipo de datos abstracto `Organizador` cuenta con la siguiente interfaz.

- `nuevo :: Organizador`
  **Propósito:** Un organizador vacío.
  **Eficiencia:** $O(1)$

- `agregarPrograma :: Organizador -> Checksum -> Set Persona -> Organizador`
  **Propósito:** Agrega al organizador un programa con el Checksum indicado; el conjunto es el conjunto de personas autores de dicho programa.
  **Precondición:** el identificador del programa que se agrega no fue usado previamente en el organizador, y el Set de personas no está vacío.
  **Eficiencia:** no hay ninguna garantía de eficiencia.

- `todosLosProgramas :: Organizador -> [Checksum]`
  **Propósito:** denota una lista con todos y cada uno de los códigos identificadores de programas del organizador.
  **Eficiencia:** $O(C)$ en peor caso, donde $C$ es la cantidad de códigos en el organizador.

- `autoresDe :: Organizador -> Checksum -> Set Persona`
  **Propósito:** denota el conjunto de autores que aparecen en un programa determinado.
  **Precondición:** el Checksum debe corresponder a un programa del organizador.
  **Eficiencia:** $O(\log C)$ en peor caso, donde $C$ es la cantidad total de programas del organizador.

- `programasDe :: Organizador -> Persona -> Set Checksum`
  **Propósito:** denota el conjunto de programas en los que participó una determinada persona.
  **Precondición:** la persona debe existir en el organizador.
  **Eficiencia:** $O(\log P)$ en peor caso, donde $P$ es la cantidad total de personas del organizador.

- `programaronJuntas :: Organizador -> Persona -> Persona -> Bool`
  **Propósito:** dado un organizador y dos personas, denota verdadero si ambas son autores de algún software en común.
  **Precondición:** las personas deben ser distintas.
  **Eficiencia:** $O(\log P + C \log C)$ en peor caso, donde $P$ es la cantidad de personas distintas que aparecen en todos los programas del organizador, y $C$ la cantidad total de programas.

- `nroProgramasDePersona :: Organizador -> Persona -> Int`
  **Propósito:** dado un organizador y una persona, denota la cantidad de programas distintos en los que aparece.
  **Eficiencia:** $O(\log P)$ en peor caso, donde $P$ es la cantidad de personas del organizador.

# Ejercicios

**Recordatorio:** De existir, agregue las precondiciones en las funciones solicitadas. ¡No deje de dividir en subtareas! Y no olvide además incluir propósito y precondiciones de las funciones auxiliares que necesite programar.

**a)** Implementar las siguientes funciones como usuario del TAD `Organizador`, establecer su eficiencia y justificarla:

*a*) `programasEnComun :: Persona -> Persona -> Organizador -> Set Checksum`
**Propósito:** dadas dos personas y un organizador, denota el conjunto de aquellos programas en las que las personas programaron juntas.

*b*) `esUnGranHacker :: Organizador -> Persona -> Bool`
**Propósito:** denota verdadero si la persona indicada aparece como autor de todos los programas del organizador.

**b)** Implementar el TAD `Organizador` suponiendo el siguiente tipo de representación:

```
data Organizador = MkO (Map Checksum (Set Persona)) (Map Persona (Set Checksum))
```

*a*) Escribir los invariantes de representación para poder crear elementos válidos del TAD.

*b*) Implementar las funciones de la interfaz, respetando las restricciones de eficiencia pedidas. Justifique en cada caso por qué se obtiene la eficiencia buscada.

**c)** Implementar una variante del TAD `Organizador` suponiendo que en la interfaz del TAD `Organizador` se agrega una nueva operación:

- `elMayorPrograma :: Organizador -> Maybe Checksum`
  **Propósito:** recibe un organizador y denota uno de los programas con más autores de todo ese organizador; denota `Nothing` si no puede devolver un programa.
  **Eficiencia:** $O(1)$ en peor caso.

Esto puede requerir modificar el tipo de representación, agregar invariantes, y modificar operaciones existentes. *Reescribir sólo las operaciones que tienen cambios sustanciales y no en las que, por ejemplo, sólo se modifica un pattern matching.*

**Recordatorio:** las interfaces de los TADs `Set` y `Map`.

La interfaz de `Set`, siendo $N$ la cantidad de elementos del conjunto:

| | |
|---|---|
| `emptyS :: Set a` | $O(1)$ |
| `isEmptyS :: Set a -> Bool` | $O(1)$ |
| `addS :: a -> Set a -> Set a` | $O(\log N)$ |
| `belongs :: a -> Set a -> Bool` | $O(\log N)$ |
| `union :: Set a -> Set a -> Set a` | $O(N \log N)$ |
| `intersection :: Set a -> Set a -> Set a` | $O(N \log N)$ |
| `set2list :: Set a -> [a]` | $O(N)$ |
| `sizeS :: Set a -> Int` | $O(1)$ |

La interfaz de `Map`, siendo $M$ la cantidad de claves distintas en el map:

| | |
|---|---|
| `emptyM :: Map k v` | $O(1)$ |
| `assocM :: k -> v -> Map k v -> Map k v` | $O(\log M)$ |
| `lookupM :: Map k v -> k -> Maybe v` | $O(\log M)$ |
| `domM :: Map k v -> [k]` | $O(M)$ |